Санкт–Петербургский государственный университет

Щербаков Глеб Александрович

Выпускная квалификационная работа

Разработка мобильного приложения, учитывающего предпочтения пользователя

Уровень образования: бакалавриат
Направление 02.03.02 «Программирование и информационные технологии»
Основная образовательная программа СВ.5002.2016 «Фундаментальная информатика и информационные технологии»

Научный руководитель:
доктор физ.-мат. наук, профессор,
кафедра моделирования экономических систем
Смирнов Николай Васильевич

Рецензент:
старший преподаватель,
кафедра технологий программирования
Стученков Александр Борисович

Санкт-Петербург
2020 г.

Содержание

Введение. Обзор литературы

К 2025 году, по прогнозам аналитической фирмы IDC [1], объем данных, находящихся на информационных носителях, во всем мире увеличится в десять раз по сравнению с 2016 годом. Уже сейчас размер информации, которую предоставляет интернет безграничен. Из-за обилия вариантов выбора пользователю очень сложно определиться с его желаниями. Поэтому помочь ориентироваться в информационном потоке очень важно. Одним из решений данной проблемы может стать аксиоматическая теория выбора, учитывающая предпочтения пользователя. Основные исследования в данной области были произведены её основателем В.Д. Ногиным [4], а также О.В. Басковым [5].

Данная теория может использоваться в любых областях: выбор плана страхования, выбор работника для найма и даже выбор фильма для просмотра. Последнее мы и используем в качестве демонстрации возможностей этого подхода. Создадим приложение, которое будет рекомендовать фильм к просмотру на основе предпочтений пользователя.

С 2019 года количество мобильных устройств, используемых в мире, превысило число персональных компьютеров и не прекращает расти [2]. Также на текущий момент более 70% мобильных телефонов работают под управлением операционной системы Android [3]. На основе этих фактов для демонстрационного приложения в качестве основной платформы выбраны мобильные устройства с Android.

Для продуктивной разработки мобильного приложения будем придерживаться современных тенденций в этой области. При реализации будут использованы принципы Clean Architecture [31], а также концепции Modulization [20]. Языком программирования приложения под Android будет Kotlin, как официально рекомендуемый компанией Google.

Наибольшую популярность при математических исследованиях получил язык Python. В связи с этим ключевой алгоритм, учитывающий предпочтения пользователя, мы реализуем с использованием данного языка. Поскольку REST[34] сервер будет использовать этот алгоритм, то процесс разработки будет проще, если он также будет реализован с использованием языка Python. В качестве библиотеки для создание сервера выступает Flask [35].

Глава 1. Предварительные сведения из теории выбора

В данной главе будут рассмотрены основные сведения из аксиоматической теории выбора, которые нам понадобятся в дальнейшем. Также будет представлена основная идея метода, который будет положен в основу нашего приложения.

1.1 Аксиоматический подход

Приведём основные понятия теории многокритериального выбора, используемые в данной работе. Данная теория была развита в основном в работах В.Д. Ногина и, в последующем, О.В. Баскова. Дальнейший обзор приводится по монографии [4].

Одной из важнейшей частей теории многокритериального выбора является лицо принимающее решение (ЛПР). Ведь процесс выбора не возможен без наличия того, кто делает этот выбор. ЛПР может быть как человек, так и целый коллектив. К сожалению, несмотря на частое упоминания ЛПР в различных математических теориях, строгого определения не существует.

Центральной конструкцией теории многокритериальной выбора является тройка $< X, f, \succ_X >$. Рассмотрим каждый из элементов подробнее.

Прежде всего должен быть дан набор вариантов (решений), из которого будет осуществляться выбор. При этом природа решений не важна, это может быть план страхования, проектное решение, фильм и т.п. Введём данное множество:

Определение 1. X *— множество возможных решений.*

Обозначим множество решений, которое выбрало ЛПР из множества возможных через $C(X)$.

Определение 2. $C(X)$ *— множество выбираемых решений.* $C(X) \subset X$.

При выборе некоторого решения ЛПР должно полагаться на какие-то параметры, на основе которых оно выбирает наиболее подходящее: самый красивый, самый производительный, самый современный и т.д. Желание ЛПР

достичь предпочитаемого результата в математических терминах можно выразить, как максимизацию (или минимизацию) некоторых числовых функций, заданных на множестве X, назовём их критериями.

Определение 3. *$g : X \to R$ — числовая функция заданная на множестве X называется критерием.*

Определение 4. *$f = (f_1, ..., f_n)$ — векторный критерий, где f_i — критерий на множестве X, $i \in \overline{1, n}$.*

Определение 5. *$I = \{1, \ldots, n\}$ — множество номеров критериев.*

Значения, которые принимают элементы из множества возможных решений X под действием векторного критерия f, называются возможными векторами.

Определение 6. *Y — множество возможных векторов, где $Y = f(X) = \{y \in \mathbb{R}^n | y = f(x)$ при некотором $x \in X\}$.*

Определение 7. *$C(Y)$ – множество выбираемых векторов, где $C(Y) = f(C(X)) = \{y \in Y | y = f(x)$ при некотором $x \in C(X)\}$.*

Теперь у нас есть множество возможных решений X и векторный критерий f. Чтобы формализовать предпочтение ЛПР нам потребуется терминология бинарных отношений, которая следует далее.

Определение 8. *$\mathfrak{R} \subset A \times B$ — бинарное отношение на множествах A и B, где $A \times B = \{(a, b) |\ a \in A, b \in B\}$. Если $A = B$, то будем говорить, что бинарное отношение $\mathfrak{R}$ задано на множестве A.*

Свойство 1. *Бинарное отношение $\mathfrak{R}$, заданное на множестве A, называют иррефлексивным, если $\nexists a \in A : a\mathfrak{R}a$.*

Свойство 2. *Бинарное отношение $\mathfrak{R}$, заданное на множестве A, называют асимметричным, если $\forall a, b \in A : a\mathfrak{R}b$, $b\mathfrak{R}a \Rightarrow a = b$.*

Свойство 3. *Бинарное отношение $\mathfrak{R}$, заданное на множестве A, называют транзитивным, если $\forall a, b, c \in A : a\mathfrak{R}b$, $b\mathfrak{R}c \Rightarrow a\mathfrak{R}c$.*

Определение 9. *Бинарное отношение $\mathfrak{R}$, заданное на множестве A, называют отношением строгого порядка, если оно иррефликсивно и транзитивно.*

Лемма 1. *Любое отношение строгого порядка является асимметричным.*

Несмотря на аксиоматический подход, мы стремимся быть близки к естественному поведению ЛПР. Поэтому наше бинарное отношение должно быть иррефлексивным, ведь нам неинтересно сравнивать вещь саму с собой. Отношение должно быть транзитивным так, как если вещь 1 лучше вещи 2, а вещь 2 лучше, чем вещь 3, то очевидно, что вещь 1 лучше вещи 3. Также естественно, что если вещь 1 лучше вещи 2, то вещь 2 не может быть лучше вещи 1. Следовательно правилом, по которому ЛПР может сравнивать два возможных решения на множестве X, будем считать бинарное отношение строго порядка.

Определение 10. *$\succ_X$ — бинарное отношение строгого предпочтения на множестве X — или просто отношение предпочтения.*

Теперь можно сформулировать постановку основной задачи многокритериального выбора. Она включает в себя:

- множество возможных решений X;
- векторный критерий f;
- отношение предпочтения $\succ_X$.

Также можно сформулировать задачу в терминах векторов. Тогда она содержит два элемента:

- множество возможных векторов $Y \subset R^m$;
- отношение предпочтения $\succ_Y$.

Ещё одним естественный предположением является аксиома Парето.

Аксиома (Парето)**.** *Для $\forall x', x'' \in X$ для которых имеет место неравенство $f(x') \geq f(x'')$, справедливо $x' \succ_X x''$.*

Следует отметить, что не из любой пары возможных решений ЛПР может сделать выбор: не любые возможные решения $x \in X$ и $z \in X$ связаны отношением $x \succ_X z$ или $z \succ_X x$. Другими словами отношение $\succ_X$ не является полным.

Свойство 4. *Бинарное отношение $\mathfrak{R}$, заданное на множестве A, называют полным, если $\forall a, b \in A$ выполняется соотношение $a\mathfrak{R}a$ или $b\mathfrak{R}a$.*

Свойство 5. *Бинарное отношение $\mathfrak{R}$, заданное на множестве A, называют частичным, если $\exists a, b \in A$ для которых ни $a\mathfrak{R}b$ ни $b\mathfrak{R}a$ не выполняются.*

Определение 11. *вектор $a \in A$ называется несравнимым относительно бинарном отношении $\mathfrak{R}$, если $\exists b \in A$ такой что, ни $a\mathfrak{R}b$, ни $b\mathfrak{R}a$ не выполняются.*

Существование несравнимых решений подводит нас к необходимости устранения этой проблемы. Для этого мы будем использовать определения квантов информации об отношении предпочтения ЛПР и соответствующую теорему.

Определение 12. *Пусть $A, B \subset I, A \neq \varnothing, B \neq \varnothing, A \cap B = \varnothing$. Будем говорить, что задан квант информации об отношении предпочтения ЛПР с двумя заданными группами критериев A и B вместе с набором положительных параметров ω_i,для $\forall i \in A$ и для $\forall j \in B$, если для $\forall y', y'' \in \mathbb{R}^m$, для которых выполняется:*

$$y'_i - y''_i = \omega_i > 0, \quad \forall i \in A,$$

$$y''_j - y'_j = \omega_j > 0, \quad \forall j \in B,$$

$$y'_s = y''_s \quad , \forall s \in I \backslash (A \cup B),$$

имеет место $y' \succ y''$.

Чтобы наше отношение предпочтения было максимально разумно и близко к тому, как ЛПР принимает решение в жизни, необходимо ввести следующие аксиомы.

Аксиома 1 (исключения доминируемый векторов)**.** *Если для некоторой пары вариантов* $y', y'' \in Y$ *выполнено соотношение* $y' \succ_Y y''$*, то* $y'' \notin C(Y)$.

Аксиома 2 (о продолжении)**.** *Существует иррефлексивное и транзитивное продолжение* $\succ_m$ *отношения предпочтения* $\succ_Y$ *на все пространство* $\mathbb{R}^m$.

Аксиома 3 (согласованности)**.** *Каждый критерий* $f_1, ..., f_m$ *согласован с отношением* $\succ_m$ *в том смысле, что если два вектора* y', y'' *отличаются по единственной компоненте* i*, причём* $y'_i > y''_i$*, то* $y' \succ_Y y''$.

Аксиома 4 (инвариантности отношения предпочтения)**.** *Отношение* $\succ_m$ *инвариантно, т.е. для* $\forall \alpha > 0, c \in \mathbb{R}^m$*, верно* $y' \succ_m y'' \Rightarrow \alpha y' + c \succ_m \alpha y'' + c$.

Сузить множество возможных решений помогает принцип Эджворта - Парето.

Определение 13. *Множество парето-оптимальных решений:*
$P_f(X) = \{x^* \in X | \nexists x \in X : f(x) \geq f(x)\}$.

Определение 14 (Принцип Эджворта-Парето)**.** *При выполнение аксиомы исключения и Парето для любого множества выбираемых решений* $C(x)$ *справедливо* $C(X) \subset P_f(X)$.

Однако мы хотим найти более точное приближение. В этом нам поспособствует следующая теорема.

Теорема 1 [5]. *Пусть выполнены аксиомы 1–4 и задан квант* y'*, несущий информацию об относительной важности групп критериев* A *и* B*. Тогда для любого множества выбираемых векторов* $C(Y)$ *верно* $C(Y) \subseteq P'(Y) \subseteq P(Y)$*, где* $P'(Y) = f(P_g(X))$*, а* g *— новый векторный критерий размерности* $p = m - |B| + |A||B|$*, компоненты которого суть* $g_{ij} = \omega_j f_i + \omega_i f_j$, $\forall i \in A, j \in B$.

Теперь у нас достаточно информации, чтобы математически строго сформулировать постановку задачи.

1.2 Общая постановка задачи

Пользователь хочет посмотреть фильм. Необходимо предоставить пользователю множество фильмов, отсортированных с учётом его предпочтений.

Опишем задачу строго математически.

За множество возможных решений X примем некоторое множество фильмов. ЛПР будем считать пользователя приложения.

Введём векторный критерий $f = (f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10})$, где

- f_1 — критерий боевика;
- f_2 — критерий вестерна;
- f_3 — критерий детектива;
- f_4 — критерий исторического фильма;
- f_5 — критерий комедии;
- f_6 — критерий триллера;
- f_7 — критерий фантастики;
- f_8 — критерий ужасов;
- f_9 — критерий драмы;
- f_{10} — критерий приключенческих фильмов.

Определение 15. *За возможные векторы фильмов возьмём результат агрегирования пользовательских рецензий к фильмам. А именно:*

W_i *— множество слов в рецензии пользователя* k *к фильму. Одному пользователю может принадлежать только одна рецензия к фильму.*

A_{f_i} *— множество слов, ассоциируемых с критерием* f_i*.*

Будем считать, что пользовательская рецензия k*, а следовательно и фильм, обладает критерием* f_i*, если* $\exists a \in A_{f_i} : a \in W_k$*.*

n_{f_i} *— количество пользовательских рецензий, которые мы отнесли к критерию* f_i*.*

n *— количество пользовательских рецензий на фильм.*

$v_i = \frac{n_{f_i}}{n} \cdot 100$ *— процент пользовательских рецензий, по которым мы определили, что критерий* f_i *присутствует в фильме.*

Возможным вектором фильма будем считать вектор $v = (v_1, \ldots, v_{10})$

$f_i(X) \in [0, 100], \quad i \in \overline{1, 10}$

В качестве отношения предпочтения примем отношение «покоординатного сравнение векторов», которое является отношением строго порядка: $(x_1, \ldots, x_n) < (y_1, \ldots, y_n)$, тогда и только тогда, когда $\forall i$ либо $x_i = y_i$ либо $x_i < y_i$.

Тогда в качестве результата предполагается получить отношение предпочтения $\succ_{P'(Y)}$, которое будет полным. $P'(Y)$ — множество возможных векторов полученных после применения теоремы 1. Теперь с помощью отношения $\succ_{P'(Y)}$ сортируем векторы в порядке убывания. Получаем отсортированный список фильмов.

1.3 План исследования

Общий план исследования можно разделить на три этапа:

На первом этапе предполагается предобработка данных с интернет-ресурса `IMDb` [12] для выделения множества возможных векторов:

- выбор 200 лучших фильмов с `IMDb` [12], выделение 100 рецензий по каждому фильму;
- получение возможных векторов фильмов.

На втором этапе реализуем основной программный комплекс:

- основной модуль, базирующийся на теореме об общих квантах;
- мобильное приложение в качестве `UI` [10];
- `REST` [34] сервер.

На третьем этапе проверяем адекватность разработанной системы:

- предоставляем инструкцию использования мобильного приложения;
- проводим опрос удовлетворённости пользователей результатами рекомендаций по фильмам.

1.4 Выводы

В данной главе мы изучили необходимые сведения из аксиоматической теории выбора. Пользуясь этим аппаратом строго сформулировали с математической точки зрения постановку задачи. И кратко описали план её реализации. Следующая глава посвящена описанию программного комплекса.

Глава 2. Программная реализация

Данная глава описывает архитектурные и программные решения, которые были приняты при создании демонстрационного программного комплекса.

2.1 Web scraper

Причина написания web-scraper [11] заключается в том, что API [33] IMDb [12] не предоставляет достаточной информации о фильме для построения возможных векторов по ним. Программа реализована на языке Python.

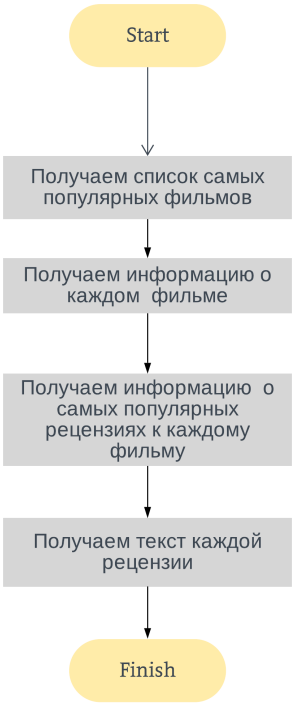

Рис. 1: Этапы работы web-scraper

На рис. 1. мы видим граф, показывающий этапы работы web-scraper.

- С помощью библиотеки BeautifulSoup [13] получаем страницу хранящую список самых популярных фильмов с IMDb. Получаем список фильмов.

- Получаем также информацию о каждом фильме: название, описание, и т.д.
- С помощью библиотеки Selenium [14] получаем список рецензий к фильму.
- Достаем текст каждой рецензии с помощью BeatifulSoup [13].

Таблица 1: Схема базы данных рецензии

id	MovieTitle	MovieReviewLink	MovieReviewChunks

Таблица 2: Схема базы данных общей информации о фильме

id	Title	Description	Year	ImageLink

Таблица 1 и таблица 2 отображают структуры таблиц заполненных с помощью парсера.

Авторская реализация расположена в репозитории [6].

2.2 Предобработка данных

Данный раздел описывает алгоритм преобразования данных, полученных с помощью веб-скрейпинга [11], в возможные векторы фильмов. Программа реализована на языке Python.

Напомним, что в качестве возможных векторов (см. определение 15) был выбран результат агрегирования пользовательский рецензий к фильмам.

С помощью библиотеки NLTK [18] создаётся словарь синонимов A_{f_i} для каждого из критериев, т.е. для критерия комедии в словарь синонимов будут добавлены следующие слова: комичный, комичесий, смешной и т.д.

Выбирается фильм. С помощью библиотеки spaCy [15] рецензия проходит через токенизацию [17], где в качестве токена выбираются существительные и прилагательные, которые к ним относятся. Далее производится

лематизация [16], в рамках которой каждое существительное и прилагательное приводится к базовой форме. Получаем множество W_k слов в рецензии.

Далее считаем n_{f_i} — количество пользователей предполагающих, что критерий f_i принадлежит фильму. Для этого смотрим, если $\exists a \in A_{f_i} : v \in W_k$, то добавляем $n_{f_i} = n_{f_i} + 1$. Составляется вектор v.

Будем считать, что начальный возможных вектор $f(x) = v$. Переходим к следующему фильму.

Таким образом мы получили множество возможных векторов. Сохраняем их в базу данных. Структура отображена в таблице 3

Таблица 3: Схема базы данных возможных векторов фильмов

Column
id
Title
Action
Western
Detective
Historical
Comedy
Thriller
Fantasy
Horror
Drama
Adventure

Таблица 4: Пример заполнения базы данных

id	Title	Ac	W	De	H	C	T	F	H	Dr	Ad
1	Начало	89	0	11	0	0	91	97	0	87	76

Таблица 4 показывает пример заполненной информации об одном фильме в базе данных.

Авторская реализация расположена в репозитории [6].

2.3 Основной алгоритм приложения

В данном разделе представлен общий алгоритм, который реализует идеи аксиоматической теории выбора. Программа реализована на языке Python. Все функции, упоминаемые в разделе, можно найти в Приложении.

На первом этапе определяется содержит ли множество возможных векторов $Y = f(X)$ несравнимые векторы (см. определение 11). Рассмотрим алгоритм определения несравнимых векторов:

1. Выбираем $v \in Y$, где $v = (v_1, \dots v_n), n = dimY$.
2. Выбираем $w \in Y$, где $w = (w_1, \dots, w_n)$.
3. Если $\exists i, j \in \overline{1, n} : sgn(w_i - v_i) \neq sgn(w_j - v_j)$ и $sgn(w_i - v_i) \neq 0$ или $sgn(w_j - v_j) \neq 0$, то v — несравнимый вектор. Добавляем v в множество Z.
4. Иначе переходим к пункту 2 и продолжаем сравнивать векторы.
5. Получаем множество несравнимых векторов Z.

Функция `get_controversial_vectors`, реализующая поиск несравнимых векторов, представлена в Приложении.

На следующем шаге из несравнимых векторов формируется множество пар $P \subset Z \times Z$ с помощью функции `get_pair_vectors`. Из каждой пары $(m, n) \in P$ пользователь выбирает один из фильмов, а следовательно и вектор, характеризующий данный фильм, он предпочитает. Если пользователю нравится фильм, связанный с вектором m, то квант информации будет равен $m - n$, иначе $n - m$. Получаем квант предпочтения. Функция, получающая множество квантов `get_quanta`, представлена в Приложении.

Теперь у нас есть n квантов $y_0^0, y_1^0, \dots, y_n^0$ важности критериев f_0. Рассмотрим k-ый шаг алгоритма, учитывающего кванты [5]:

1. На основе теоремы 1 строим матрицу T_k, преобразующую f_{k-1} в новый векторный критерий $f_k = T_k f_{k-1}$ учитывающий y_{k-1}.

2. Строим оставшиеся кванты $y_{k+1}^k = T_k y_{k+1}^{k-1}, \ldots, y_n^k = T_k y_n^{k-1}$.

3. Если среди этих векторов $y_j^k \leq 0$, то алгоритм завершается т.к. обнаружено противоречивое сообщение.

4. Отбрасываем векторы $y_j^k \geq 0$. Будем считать, что оставшиеся кванты $y_{k+1}^k, \ldots, y_n^k$.

5. Если ещё остались кванты, то переходим к следующей итерации.

6. Иначе завершаем, получаем матрицу $T = T_n T_{n-1} \ldots T_2 T_1$.

Изучить функцию, реализующую учёт квантов `get_final_matrix_criteria` , можно в Приложении.

Полученную в результате матрицу T применяем для преобразования множества возможных векторов Y. Вычисляем новое множество

$$S = \{s | s = Ty, y \in Y\}.$$

Если множество S не содержит несравнимых векторов, тогда сортируем векторы и выводим итоговый список фильмов X с учётом отношения предпочтения пользователя. Иначе возвращаемся к этапу с формированием множества пар несравнимых векторов. Функции преобразования `get_final_vectors` и сортировки `sort` представлены в Приложении.

Авторская реализация расположена в репозитории [7].

2.4 Архитектура мобильного приложения

В данном разделе будут рассмотрены архитектурные решения, которые были использованы при разработке мобильного приложения, а также используемые библиотеки. Мобильное приложение реализовано на языке Kotlin под операционную систему Android.

При разработке мобильного приложения был использован современный подход, представленный компанией Google – Modularization [20]. Данный подход позволяет избежать возникновения монолитного приложения, которое значительно замедляет разработку и тестирование. В данной работе используется улучшенная версия, разработанная компанией ЦФТ [23].

Важным понятием в Modulazition является модуль. Модули бывают четырёх типов.

Определение 16. *Модуль называется component, если он удовлетворяет следующим требованиям:*

- *содержит базовую и часто переиспользуемую логику;*
- *не зависит от других модулей.*

Определение 17. *Модуль называется shared, если он удовлетворяет следующим требованиям:*

- *зависит только от shared модуля или component модуля;*
- *содержит общий код для нескольких модулей;*
- *ограничен одной зоной ответственности.*

Определение 18. *Модуль называется feature, если он удовлетворяет следующим требованиям:*

- *не зависит от других feature модулей;*
- *содержит обособленный функционал.*

Определение 19. *Модуль называется app, если он удовлетворяет следующим требованиям:*

- *существуют в единственном экземпляре;*
- *зависит от всех модулей.*

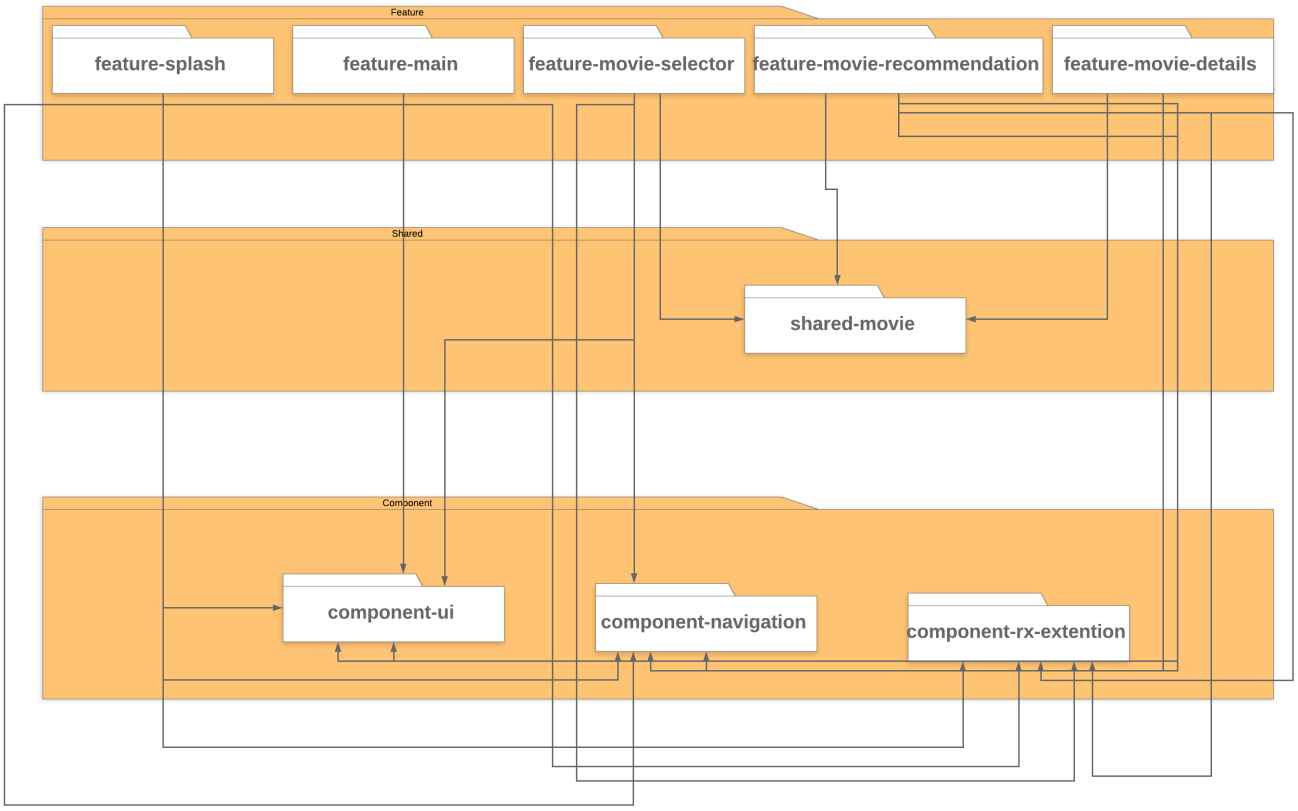

Рис. 2: Архитектура мобильного приложения

На рис. 2 представлена архитектура приложения с учётом ранее рассмотренных определений. Чтобы диаграмма зависимостей не была перегруженной связями, в изображении не отображён app модуль. Более детально распишем, что содержит каждый из модулей:

- component-navigation набор классов и интерфейсов для реализации переходов между экранами;
- component-ui содержит базовый каркас MVVM [21], общие компоненты интерфейса, расширения базовых классов таких, как Fragment [22] и т.д;
- shared-movie содержит классы объединённые доменной областью фильм [19];
- feature-splash содержит логику экрана SplashScreen;
- feature-main содержит логику экрана MainScreen;

- feature-movie-selector содержит логику экрана MovieSelectorScreen;
- feature-movie-recommendation содержит логику экрана MovieRecommendationScreen;
- feature-movie-details содержит логику экрана MovieDetailsScreen.

Библиотеки, которые использовались в мобильном приложении:

- Dagger 2 — самая популярная библиотека DI (dependency injection) [24].
- Retrofit — библиотека для реализация REST-клиента [25].
- RxJava/RxAndroid — библиотеки для реализации реактивного подхода [26].
- Glide — библиотека для загрузки и кэширования изображений [27].
- JUnit — библиотека для написания Unit тестов [28].
- Epresso — библиотека для написания UI тестов [29].
- Material, ViewPager, RecyclerView, Constrain — библиотеки Google для реализация популярных View элементов [30].

Также приложение написано по стандартам Clean Architecture [31]. Авторская реализация расположена в репозитории [8].

2.5 API REST сервера

Данный раздел описывает API [33] REST [34] сервера. Сервер реализован с использованием фреймворка Flask [35] и языка программирования Python.

```
{
  "data":[
    {
      "id":0,
      "title":"Властелин колец: Возвращение короля",
      "imageLink":"https://m.media-amazon.com/images/something.jpg",
      "criteria":{
        "боевик":86,
        "вестерн":0,
        "детектив":0,
        "исторический фильм":0,
        "комедия":0,
        "триллер":0,
        "фантастика":0,
        "ужасы":0,
        "драма":0,
        "приключенческий фильм":93
      }
    },
    {
      "id": 3
      ...
    },
    ...
  ]
}
```


Рис. 3: Пример ответа

Запрос на получения фильмов для выбора:
`GET:htttp://host/movies_for_selection`
Пример ответа рис. 3.


```
{
  "data":[
    {
      "id":0,
      "title":"Властелин колец: Возвращение короля",
      "imageLink":"https://m.media-amazon.com/images/something.jpg",
      "criteria":{
        "боевик":86,
        "вестерн":0,
        "детектив":0,
        "исторический фильм":0,
        "комедия":0,
        "триллер":0,
        "фантастика":0,
        "ужасы":0,
        "драма":0,
        "приключенческий фильм":93
      }
    },
    {
      "id": 3
      ...
    },
    ...
  ]
}
```


Рис. 4: Пример тела ответа

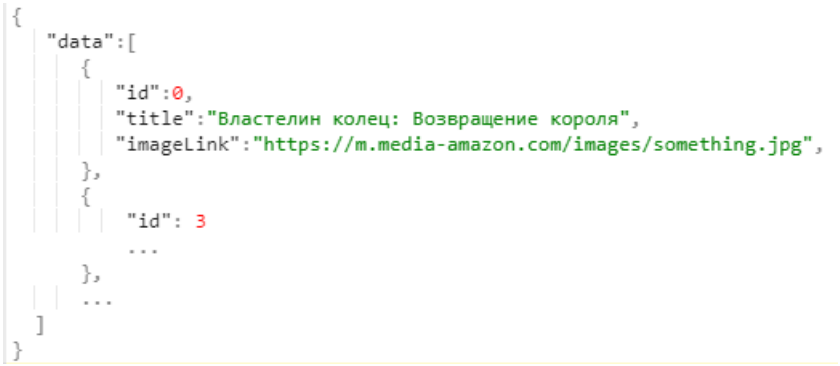

Рис. 5: Пример тела запроса

Запрос для получения рекомендованных к просмотру фильмов:

`POST:htttp://host/recommended_movies`

Пример запроса рис. 4.

Пример ответа рис. 5.

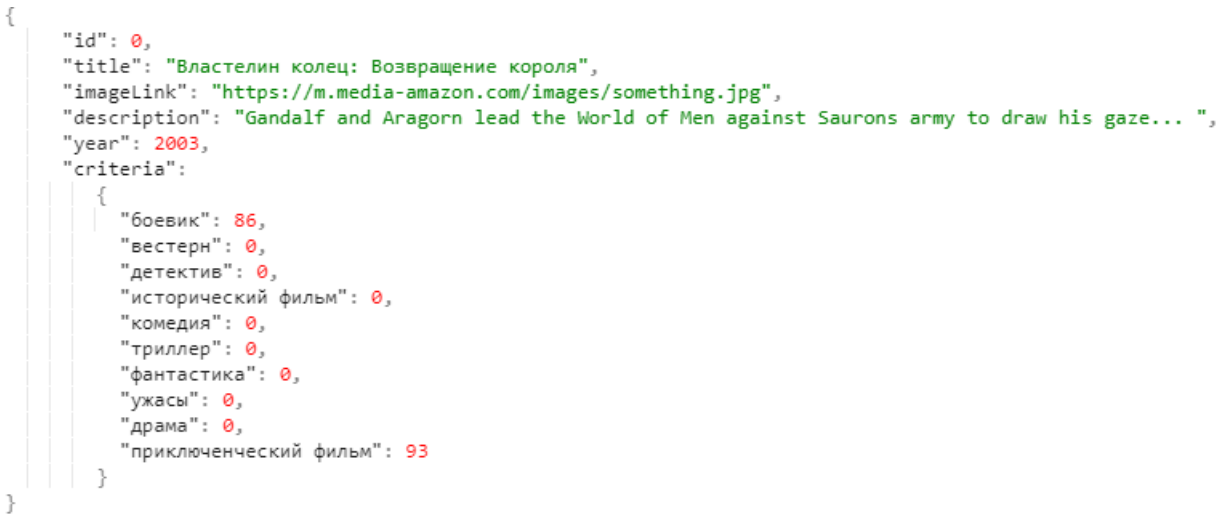

Рис. 6: Пример тела ответа

Запрос для получения подробной информации о фильме:

`GET:htttp://host/movie_details?id=`

Параметр запроса: id фильма в базе данных.

Пример ответа рис. 6.

Авторская реализация расположена в репозитории [9].

2.6 Выводы

В данной главе был описан ключевой алгоритм приложения, а также рассмотрена архитектура мобильного приложение и принципы его разработки. Представлен API [33] сервера. В процессе реализации при разработке математической части предпочтение было отдано языку Python, как получившему большую популярность при научных исследованиях. Для мобильного приложения был использован Kotlin, как официально рекомендуемый для разработки под Android.

Глава 3. Порядок использования мобильного приложения

В данной главе будут рассмотрены экраны приложения, а также представлена инструкция для его использования.

3.1 Описание экранов мобильного приложения

Данный раздел демонстрирует пользовательский интерфейс мобильного приложения, а также краткое описание функций каждого экрана.

Рис. 7: SplashScreen

На экране SplashScreen (см. рис. 7) осуществляется проверка доступности сервера и отображается логотип приложения.

Рис. 8: MainScreen

На экране MainScreen (см. рис. 8) отображаются главные функции приложения.

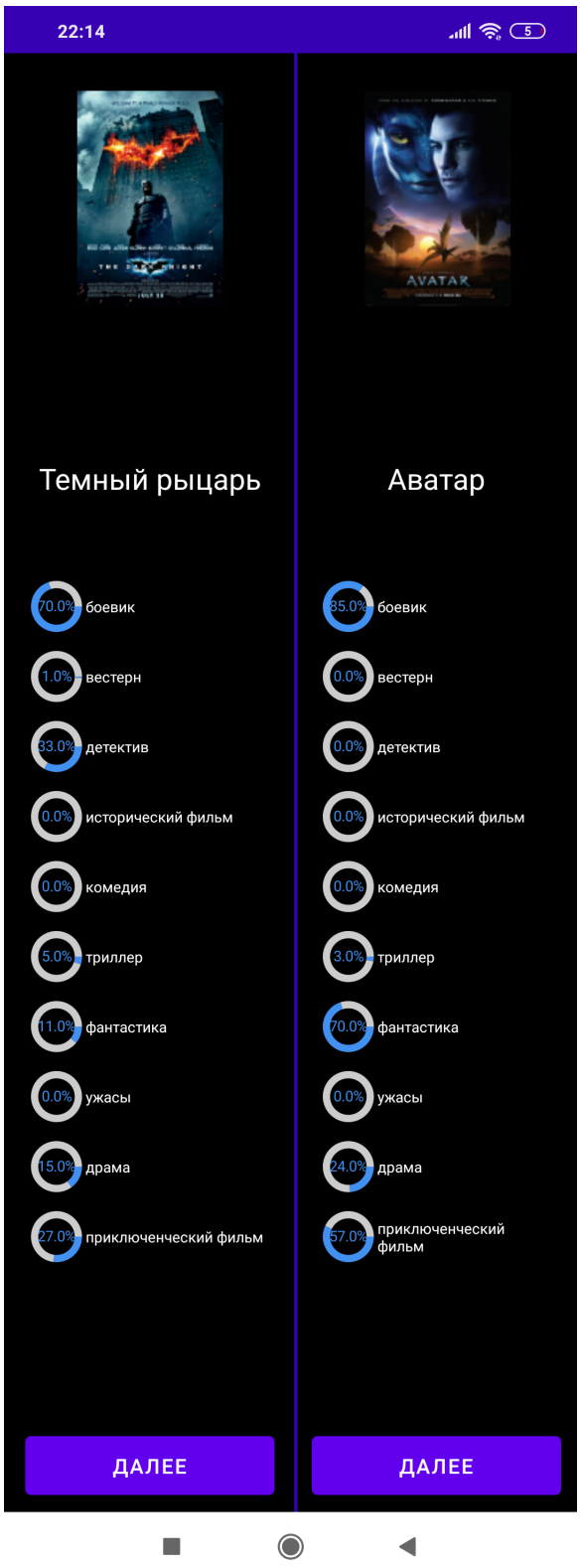

Рис. 9: MovieSelectorScreen

Экран MovieSelectorScreen (см. рис. 9) предоставляет информацию о

двух фильмах для сравнения: постер, название фильма, и список критериев.

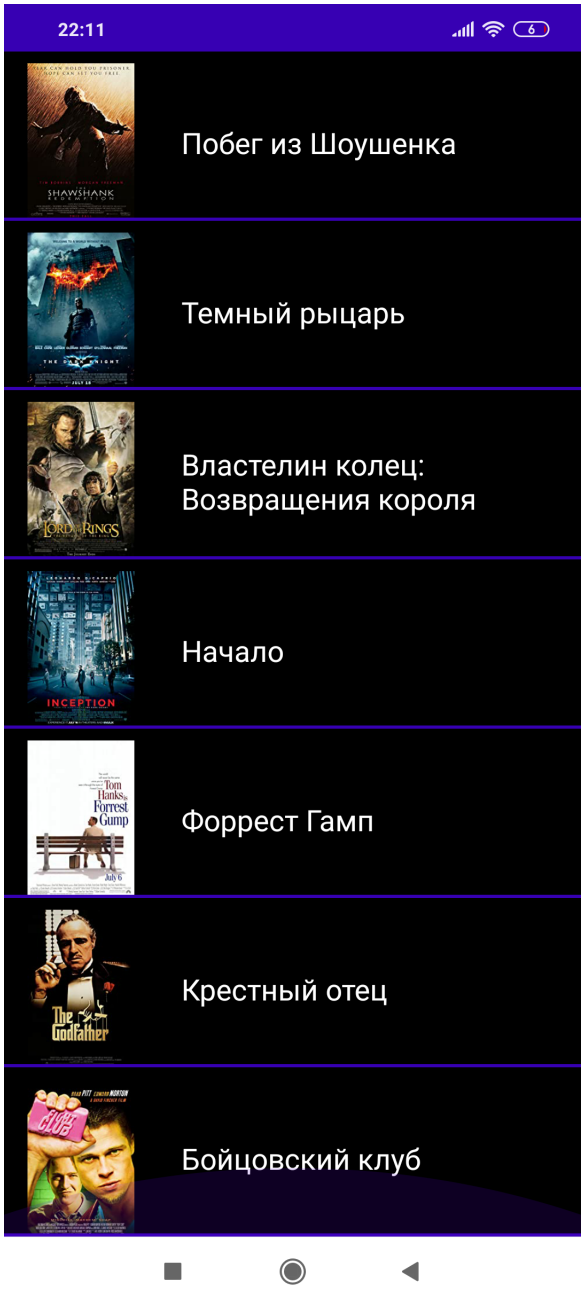

Рис. 10: MovieRecommendationScreen

На экране MovieRecomendationScreen (см. рис. 10) показываются список рекомендованных к просмотру фильмов на основе предпочтений пользователя.

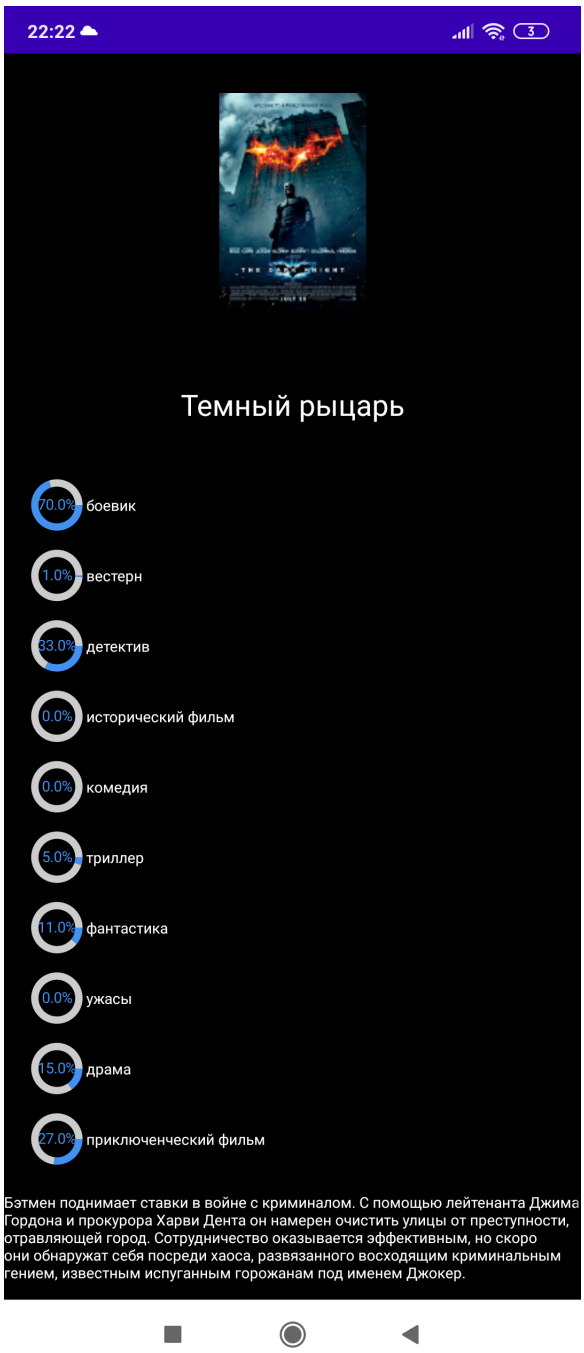

Рис. 11: MovieDetailsScreen

Экран MovieDataildsScreen (см. рис. 11) позволяет узнать больше информации о фильме для принятие решение о его просмотре.

3.2 Стандартный алгоритм использования приложения

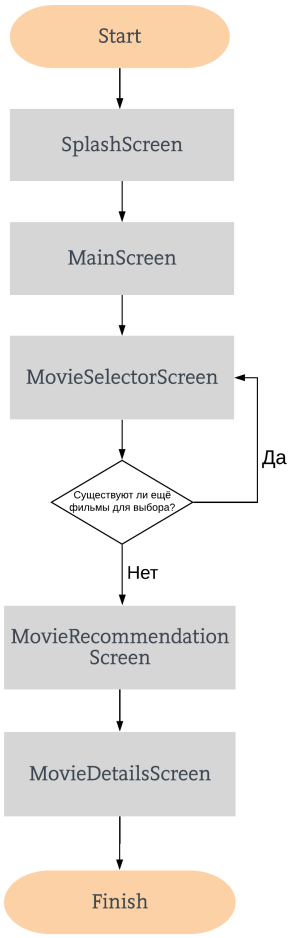

Рис. 12: Flow мобильного приложения.

На рис. 12 представлен граф переходов[32], описывающий работу мобильного приложения. А именно:

1. Пользователь запускает приложение.
2. На экране MainScreen пользователь нажимает на кнопку «Что посмотреть» и переходит на MovieSelectorScreen.
3. На MovieSelectorScreen пользователь выбирает какой из двух предоставленных фильмом он бы предпочёл посмотреть.
4. Если существуют ещё фильмы для выбора, то возвращаем к п.3, иначе переходим на экран MovieRecommendedScreen.

5. На экране MovieRecommendedScreen пользователь выбирает понравившийся фильм из списка фильмов, рекомендованных на основе его выбора, и переходит на экран MovieDetailsScreen.

6. На экране MovieDetailesScreen пользователь узнает более подробную информацию о фильме.

3.3 Выводы

Итоговое приложение было предложено для тестирования 14 людям. Пользователи подтвердили высокую точность работы приложения: порядок следования списка рекомендованных фильмов совпадал с их предпочтениями. На основе полученных результатов можно прийти к выводу об успешности использования аксиоматической теории выбора в разработке программных средств, предсказывающих желания пользователя.

Заключение

К основным результатам работы можно отнести:

- Изучена аксиоматическая теория выбора и её возможности для предсказания предпочтений пользователя.
- Создан общий модуль на основе теоремы 1, позволяющий использовать его для разработки подобных приложений в других областях.
- Реализовано мобильное приложение, как демонстрационный стенд, отражающий потенциал использования аксиоматической теории при разработке приложений.

Список литературы

[1] IDC [Электронный ресурс]: URL:https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf (дата обращения: 20.02.20).

[2] GlobalStats [Электронный ресурс]: URL:https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet (дата обращения: 20.02.20).

[3] GlobalStats [Электронный ресурс]: URL:https://gs.statcounter.com/os-market-share/mobile/worldwide (дата обращения: 20.02.20).

[4] Ногин В. Д. Сужение множества Парето. М.: ФИЗМАТЛИТ, 2016.

[5] Басков О. В. Труды XLI Международной научной конференции аспирантов и студентов. СПб.: СПбГУ, 2010, 553 с.

[6] Щербаков Г. А. Репозиторий ImdbPareser [Электронный ресурс]: URL:https://github.com/igamgam97/ImdbParser (дата обращения: 29.05.20).

[7] Щербаков Г. А. Репозиторий CorePreferenceAccountingModule [Электронный ресурс]: URL:https://github.com/igamgam97/CorePreferenceAccountingModule (дата обращения: 29.05.20).

[8] Щербаков Г. А. Репозиторий WhatMeToWatch [Электронный ресурс]: URL:https://github.com/igamgam97/WhatMeToWatch (дата обращения: 29.05.20).

[9] Щербаков Г. А. Репозиторий RecommendedMovieServer [Электронный ресурс]: URL:https://github.com/igamgam97/RecommendedMovieServer (дата обращения: 29.05.20).

[10] Wikipedia [Электронный ресурс]: URL:https://en.wikipedia.org/wiki/User_interface (дата обращения: 20.02.20).

[11] Wikipedia [Электронный ресурс]: URL:https://en.wikipedia.org/wiki/Web_scraping (дата обращения: 4.03.20).

[12] Wikipedia [Электронный ресурс]: URL:https://en.wikipedia.org/wiki/IMDb (дата обращения: 4.03.20).

[13] GitHub [Электронный ресурс]: URL:https://github.com/wention/BeautifulSoup4 (дата обращения: 4.03.20).

[14] GitHub [Электронный ресурс]: URL:https://github.com/SeleniumHQ/selenium (дата обращения: 4.03.20).

[15] GitHub [Электронный ресурс]: URL:https://github.com/explosion/spaCy (дата обращения: 4.04.20).

[16] Wikipedia [Электронный ресурс]: URL:https://en.wikipedia.org/wiki/Lemmatisation (дата обращения: 4.04.20).

[17] Wikipedia [Электронный ресурс]: URL:https://en.wikipedia.org/wiki/Lexical_analysis#Tokenization (дата обращения: 4.04.20).

[18] GitHub [Электронный ресурс]: URL:https://github.com/nltk/nltk (дата обращения: 4.04.20).

[19] Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. USA.: Addison-Wesley Professional, 2003.

[20] Medium [Электронный ресурс]: URL:https://medium.com/swlh/modularization-by-feature-and-layer-with-android-architecture-com (дата обращения: 20.02.20).

[21] Microsoft [Электронный ресурс]: URL:https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)?redirectedfrom=MSDN (дата обращения: 20.02.20).

[22] Google Android Developer [Электронный ресурс]: URL:https://developer.android.com/guide/components/fragments (дата обращения: 20.02.20).

[23] GitHub [Электронный ресурс]: URL:https://www.cft.ru/ (дата обращения: 20.02.20).

[24] GitHub [Электронный ресурс]: URL:https://github.com/google/dagger (дата обращения: 20.02.20).

[25] GitHub [Электронный ресурс]: URL:https://github.com/square/retrofit (дата обращения: 20.02.20).

[26] GitHub [Электронный ресурс]: URL:https://github.com/ReactiveX/RxJava (дата обращения: 20.02.20).

[27] GitHub [Электронный ресурс]: https://github.com/bumptech/glide (дата обращения: 20.02.20).

[28] GitHub [Электронный ресурс]: https://github.com/junit-team/junit5 (дата обращения: 20.02.20).

[29] GitHub [Электронный ресурс]: https://github.com/android/android-test (дата обращения: 20.02.20).

[30] GitHub [Электронный ресурс]: https://github.com/material-components/material-components-android (дата обращения: 20.02.20).

[31] Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. USA.: Prentice Hall, 2017.

[32] Wikipedia [Электронный ресурс]: URL:https://en.wikipedia.org/wiki/Flowchart (дата обращения: 25.04.20).

[33] Wikipedia [Электронный ресурс]: URL:https://en.wikipedia.org/wiki/Application_programming_interface (дата обращения: 20.04.20).

[34] Wikipedia [Электронный ресурс]: URL:https://en.wikipedia.org/wiki/Representational_state_transfer (дата обращения: 20.04.20).

[35] Github [Электронный ресурс]: URL:https://github.com/pallets/flask (дата обращения: 20.04.20).

Приложение

Здесь приведён код основного алгоритма приложения. Все функции написаны на языке Python.

Код формирования пар возможных вектор для создания квантов информации

Данная функция формирует из множества векторов множество пар, предоставляемых пользователю для выбора.

```
def get_pair_vectors(matrix, number_pairs=4):
    rng = default_rng()
    movie_index = rng.choice(matrix.shape[0], size=number_pairs * 2, replace=False)
    return matrix[movie_index, :], movie_index
```

Код нахождения квантов

Данная функция соответствует алгоритму вычисления квантов информации, представленному в разделе 2.3.

```
def get_quanta(matrix, vector_preferences):
    quanta = np.zeros((vector_preferences.shape[0], matrix.shape[1]))
    index = 0
    while index < vector_preferences.shape[0]:
        quantum = get_quantum(matrix[2 * index], matrix[2 * index + 1], \
              vector_preferences[index])
        quanta[index] = quantum
        index = index + 1
    return quanta

def get_quantum(first_vector, second_vector, first_prefer=True):
    if first_prefer:
        quantum = first_vector - second_vector
    else:
        quantum = second_vector - first_vector

    return quantum
```

Код нахождения несравнимых векторов

Данная функция соответствует алгоритму поиска несравнимых векторов, представленному в разделе 2.3.

```
def get_controversial_vectors(matrix):
    number_rows = int(matrix.shape[0])
    list_index_controversial_vectors = []
    for vector_index in range(0, number_rows):
        if is_controversial_vector(vector_index, matrix):
            list_index_controversial_vectors.append(vector_index)
    return list_index_controversial_vectors

def is_controversial_vector(vector_index, matrix):
    vector = matrix[vector_index, :]
    matrix = np.delete(matrix, vector_index, axis=0)
    number_columns = int(matrix.shape[1])
    for other_vector in matrix:
        previous_sign_elem = 0
        sign_elem = 0
        for index_elem in range(0, number_columns):
            if vector[index_elem] > other_vector[index_elem]:
                sign_elem = 1
            elif vector[index_elem] < other_vector[index_elem]:
            sign_elem = -1
            else:
                continue
            if abs(sign_elem - previous_sign_elem) == 2:
                return True
            previous_sign_elem = sign_elem
    return False
```

Код учёта квантов информации

Данная функция соответствует алгоритму учёта квантов информации, представленному в разделе 2.3.

```
def get_final_matrix_criteria(quantum_list):

    final_matrix_criteria = np.eye(quantum_list.shape[1])

    while quantum_list.shape[0] > 0:
        quantum = quantum_list[0]
        T = get_matrix_criteria_by_quantum(quantum)
```

```
8           final_matrix_criteria = T.dot(final_matrix_criteria)
9           quantum_list = np.delete(quantum_list, 0, axis=0)
10          quantum_list = T.dot(quantum_list.transpose()).transpose()
11          if is_negative_vector(quantum_list):
12              raise ContradictoryQuantumException()
13          quantum_list = get_not_positive_vectors(quantum_list)
14
15      return final_matrix_criteria
16
17  def get_matrix_criteria_by_quantum(quantum):
18      A = np.argwhere(quantum > 0)
19      B = np.argwhere(quantum <= 0)
20      p = quantum.shape[0]
21      m = p - B.shape[0] + B.shape[0] * A.shape[0]
22      T = np.zeros((m, p))
23      for index, a in enumerate(A):
24          T[index, a] = 1
25
26      index = A.shape[0]
27      while index < m - A.shape[0]:
28          for a in A:
29          weight_a = quantum[a]
30          for b in B:
31              weight_b = quantum[b]
32              T[index, a] = abs(weight_b)
33              T[index, b] = abs(weight_a)
34              index += 1
35      return T
36
37  def is_negative_vector(vector):
38      if vector.size != 0 and (vector <= 0).all():
39          return True
40      else:
41          return False
42
43  def get_not_positive_vectors(matrix):
44      result = np.invert(np.all(matrix >= 0, axis=1))
45      return matrix[result, :]
```

Код получения возможных векторов с учётом квантов

Данная функция соответствует алгоритму создания нового множества S представленному в разделе 2.3, где S — множество, полученное в результате применения матрицы учёта квантов T к множеству возможных векторов.

```
def get_final_vectors(T, matrix):
    return T.dot(matrix.transpose()).transpose()
```

Сортировка возможных векторов

Данная функция сортирует вектора в порядке убывания с использованием алгоритма быстрой сортировки.

```
def sort(matrix):
    return np.argsort(-matrix, axis=0)
```